From Legacy Model to Future Federate: The Janus Simulation Object Model

Leroy A. Jackson
Operations Research Analyst
United State Army
TRADOC Analysis Center—Monterey
Monterey, CA

Arnold H. Buss
Adjunct Professor
Operations Research Department
Naval Postgraduate School
Monterey, CA

Larry R. Larimer
Operations Research Analyst
United States Army
TRADOC Analysis Center
Fort Leavenworth, KS

Keywords:
SOM, HLA, ADS, OMDT, Janus, Legacy Simulation

ABSTRACT: *The development team has been investigating the feasibility of including Janus in future HLA Federations. One necessary condition for a potential Federate is that it have an HLA Simulation Object Model (SOM). In this paper we will describe a conceptual mode and a SOM developed for Janus and the methodology used to develop it. This report provides the final results of work described in 97S-SIW-138 during the previous workshop. It includes a more detailed description of the methodology outlined in the previous paper. It highlights how the process used for SOM development differs in some critical ways from the current recommended FOM/SOM development process model. It stresses the benefits of first building the conceptual model of a legacy simulation during SOM development. It explores some advantages of including an analyst on the SOM development team.*

1.0 Introduction

The High Level Architecture (HLA) holds the promise for interoperability of simulations by their participation in HLA Federations. Ultimately, a new generation of simulations will be produced, each of whose design incorporated HLA and object model concepts. Until these simulations are fully implemented, tested, and have undergone the VV&A process, the only source for concrete, valid models is the reservoir of legacy simulations.

One such legacy simulation is Janus. This paper describes the process undergone in taking Janus through a crucial step in the HLA process, the development of a Simulation Object Model (SOM).

The process for constructing a SOM from a legacy simulation consists of first developing a Comprehensive Object Model (COM), an object model that captures all the essential features of the simulation. The COM should be sufficiently complete that a SOM may be extracted by simply removing those elements not necessary for interoperability. Conversely, the COM is an accurate, albeit abstract, object-oriented depiction of the simulation's capabilities. Consequently, the task of determining a simulation's suitability for inclusion in a federation is made easy; if examination of a simulation's COM indicates lack of necessary features, then they are likely not to be supported by the underlying simulation model.

2.0 Background

Janus is a high resolution, interactive, six-sided, closed, stochastic, ground combat simulation. Lawrence Livermore National Laboratories developed Janus to model nuclear effects, and the U.S. Army's Training and Doctrine Command (TRADOC) Analysis Center (TRAC) at White Sands Missile Range (TRAC-WSMR) is responsible for subsequent Janus development. TRAC-WSMR modified Janus extensively for Army high resolution combat model requirements. Since its fielding in 1978, Janus has been used extensively within the U.S. Army for both training and analysis. Janus is also used for analysis by RAND Corporation, the United States Marine Corps, and by the armed forces of the United Kingdom, Australia, France, and Germany.

Janus represents a substantial investment from DoD and the U.S. Army. There is considerable incentive to extend its useful life. This has resulted in numerous enhancements to Janus, extending its capabilities considerably beyond those originally envisioned. Janus has been distributed with other Janus simulations [12] and with other models [13].

2.1 Janus SOM Development Project

The Janus SOM Development Project was initiated to investigate the feasibility for Janus participation in future HLA federations. However, Janus, as a legacy model, provides some significant challenges in meeting the HLA requirements. Janus is coded in a procedural language with no well-documented object model definition. The ongoing research is to determine if Janus can be described in an HLA compatible way by developing an HLA SOM for Janus (JSOM) that is both useful for some HLA federation and is faithful to the capabilities and limitations of Janus. The success of this effort will help pave the way for other legacy models to conform to HLA requirements and to participate in future HLA federations.

Preliminary work on the JSOM Development Project was reported in [7]. Further results are reported in [8].

2.3 Janus as an Analytic Tool

Historically, Janus has been a highly successful analysis tool to research the effectiveness of new military systems and tactical doctrines. Two components are key for this success: a flexible database and a powerful post processor.

The robust representation of systems in the database allows the analyst to model new military systems and proposed modifications to existing systems. Systems are modeled as a combination of a platform, weapon systems, and sensors. The database includes nearly every ground vehicle combat system, dismounted crew-served weapon system, and Army rotary wing aircraft in the U.S. inventory and most of those used by threat nations. Systems that are not in the database can be easily created. Over 350 attributes are available in the database for the analyst to model platforms, weapon systems, sensors, projectiles, barriers, and weather. Table 1 provides the reader with a more detailed summary of the attributes in the Janus database available to model entities. The current version of Janus also represents limited types of fixed wing aircraft and precision guided munitions.

The Janus Post Processor details entity interactions that occur during the execution of each scenario. Output reports include an artillery fire report, indirect fire ammunition expenditure report, direct fire reports, detection tables, coroner's report, and killer/victim scoreboard. Additionally, Janus provides a supplementary tool, the Janus Analyst Workstation. This tool has an "instant replay" capability for viewing events graphically as they occurred during the scenario run. The Janus Analyst Workstation also provides statistical output that is synchronized with the scenario run time to assist the analyst. [5]

3.0 Methodology

3.1. Overview

The methodology used to create the Janus SOM consists of two steps, as shown in Figure 1. First, a Comprehensive Object Model (COM) is built from the simulation. The second step consists of extracting a SOM from the COM.

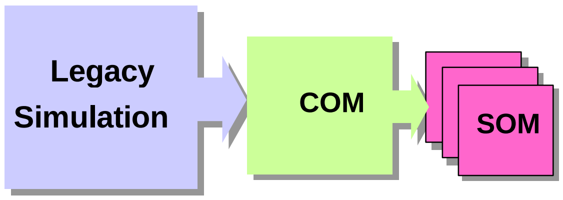

Figure 1: COM Development Process

The COM is a comprehensive, object-oriented description of the underlying simulation model, whereas an HLA SOM is more special-purpose. The COM enables an analyst to include additional information in the SOM above and beyond the minimum requirements for interoperability. Development of the Comprehensive Object Model requires a working knowledge of the simulation being modeled. In this case, the user's manual, graphic user interface, database, database manual, and software design manual provided sufficient knowledge of the simulation to produce the COM. The basic steps in the COM development process are shown in Table 1.

Table 1: Steps in the COM Process

1. Identify all the instantiable objects in the simulation.
2. Identify all the attributes used to define the instantiable objects.
3. Develop an object class structure.
4. Assign the attributes to the appropriate abstract class level for each object class.
5. Identify all possible interactions and associated interaction parameters.
6. Return to object attributes to include those attributes identified while working on interaction data and verify that new attributes do not reveal additional attributes that were previously overlooked.
7. Return to interaction table and verify all necessary attributes and parameters are noted.
8. Use above information to complete object model tables.

The steps shown in Table 1 produced the bulk of the information necessary to complete the object model tables. The component structure and associations were identified after the process outlined above was complete and before work on the tables began.

The object model lexicon and attribute parameter definitions were entered as the tables were produced using the Aegis Research Object Model Development Tool (OMDT) software [10]. The OMDT allows the user to enter object model lexicon and attribute/parameter definitions from the object class structure table. In this way, the OMDT greatly simplifies the task of entering this information and avoids the difficulty of switching frequently from attribute/parameter table to object model lexicon table and back to make appropriate entries.

After the COM is complete, it is a relatively simple matter to extract the appropriate table entries to create the HLA SOM. As discussed earlier, there may be compelling reasons to include additional information in the SOM above the minimal requirements for simulation interoperability. Additionally, should it be deemed necessary or appropriate, multiple SOMs could easily be extracted from the COM, each targeted at a specific HLA federation.

3.2. Detailed Description of Methodology

1. *Identify all the instantiable objects in the simulation.* In the case of an entity-level simulation such as Janus, this step consists of determining precisely which entities exist in the model. This step is more conceptual when the simulation is procedural (as in the case of Janus) than if the model was originally object-oriented. A model implemented in a procedural manner has no built-in notion of distinct entities, as is the case with object-oriented implementations. Consequently, the "objects" in the COM and SOM are conceptual representations of the underlying simulation rather than explicit reflections of the underlying structure.
2. *Identify all the attributes used to define the instantiable objects.* For procedural models such as Janus, this step can be the most challenging. Documentation for many attributes can be located in the model's database, the user interface, or the manuals. However, short of reviewing the actual code, it can be difficult to be certain that all relevant attributes have been identified.
3. *Develop an object class structure.* The object class structure was largely determined by observing similar classes with the potential for common attributes. Note that the class hierarchy was developed from the bottom up rather than from the top down.
4. *Assign the attributes to the appropriate abstract class level for each object class.* The common attributes identified in the previous stage were assigned to the highest possible position in the hierarchy.
5. *Identify all possible interactions and associated interaction parameters.* This step also identified some object attributes that were previously overlooked.
6. *Return to object attributes to include those attributes identified while working on interaction data and verify that new attributes do not reveal additional attributes that were previously overlooked.* This step essentially

repeats Step 5 with the new attributes identified in Step 6.

7. *Return to interaction table and verify all necessary attributes and parameters are noted.* The new attributes added in Step 6 could give rise to further interactions and interaction parameters.
8. *Use above information to complete object model tables.* When no additional interactions produce new attributes or attributes interactions, the information is assembled in the completed object model tables.

3.3. Potential Omissions

Despite the effort and thought that have gone into the JSOM development process outlined in this document, the complexity of the Janus simulation suggests the potential for errors or omissions in the completed Janus COM and SOM.

Because the approach to SOM development was conceptual, there is some chance that small numbers of object attributes and interaction parameters may have been overlooked. Another approach would have been to produce the Comprehensive Object Model and the SOM by reviewing the Janus code. There are benefits and drawbacks associated with both approaches.

There are primarily two benefits to the conceptual approach. The object model development is not restricted by the structure and implementation of the simulation code, and the time required to produce a near complete object model is vastly reduced. Without looking at the underlying code, the modeler is free to use both operational experience and knowledge of object model methodologies to produce an object oriented representation of the simulation. This facet of the conceptual approach seemed to be particularly advantageous while working with a procedurally implemented legacy simulation.

The second benefit of the conceptual approach, reduced object model development time, was also important in working with Janus. Janus consists of over two hundred thousand lines of code and includes many code improvements and upgrades incorporated over the last twenty years. While Janus is well documented, the time required to gain a working knowledge of the Janus code, and then to follow the code to produce an object representation would have been prohibitive.

The most likely errors of omission occur in the attribute/parameter table and the interaction table. The Janus simulation uses hundreds of attributes to define the many classes of objects. An overwhelming percentage of these attributes was easily identified by reviewing the database, graphic user interface, user’s manual, and software design manual. However, some attributes were identified by supposition. It is this relatively small category of attributes that indicates there may be others that were overlooked. Similarly, there may be interaction parameters that were omitted from the interaction table.

3.4 Example: The Platform Class

The initial Object Class Structure Table was constructed using an organization chart format. This simple format provided a clearly defined class hierarchy and structure for later documentation in the HLA object model template tables.

The platform subtree of this class hierarchy is based primarily on the Janus database which lists each platform the user might introduce into a scenario. Examples of these platforms include the M1A1 Abrams tank, the M2 Bradley Infantry Fighting Vehicle, and the individual rifleman. Starting with these platforms as the instantiable objects in the class hierarchy, a tentative hierarchy of abstract classes was produced by extracting common attributes for the superclasses. Ultimately, the base platform superclass was reached. The resulting object class hierarchy is depicted inFigure 1.

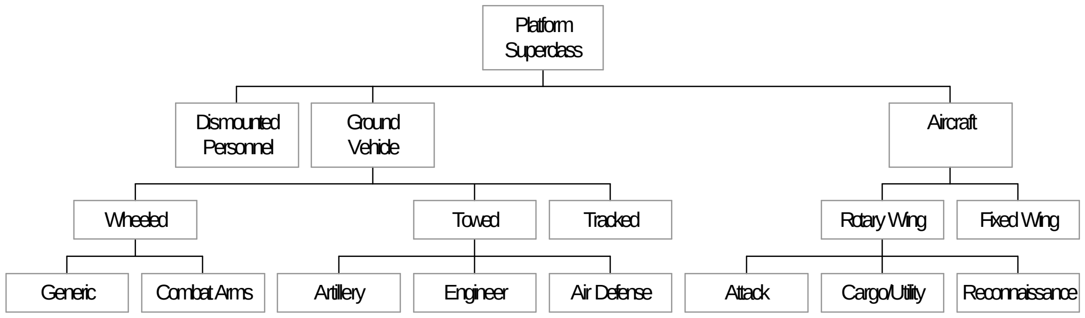

Figure 2: Original Platform Class Hierarchy

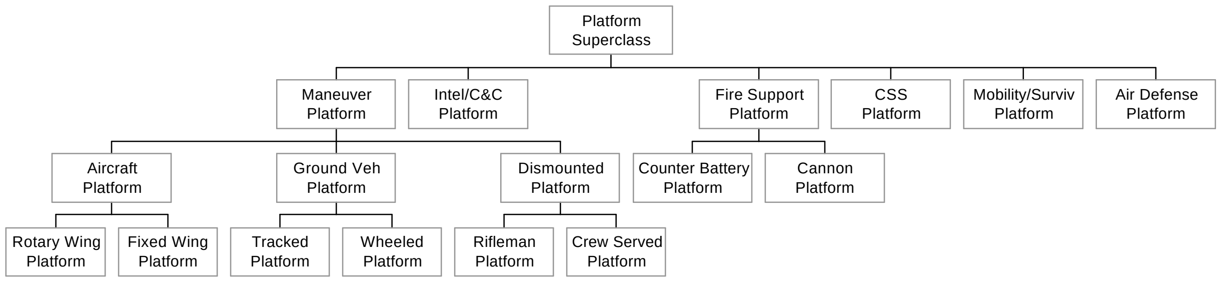

Figure 3: Revised Platform Class Hierarchy

The class hierarchy in Figure 2 corresponds closely with conventional hierarchies developed for platforms. Although constructed from the bottom up, as described above, the decision to first abstract functional attributes at the early stages produced a hierarchy in which the base platform superclass is first subclassed by physical type (ground vehicle vice aircraft, for example) with the functional distinctions appearing lower in the tree. This structure is most useful for federations in which the physical distinctions are more important to the other federates, since subscription to the platform types may be done at a relatively high level. However, for federations in which the concern of the federates is more at the functional level, this structure is less useful. Subscription must be done at lower levels in the hierarchy, and it is more difficult for a fellow federate to determine subscription requirements. An example of the latter type of federation is the Analysis Federation proposed by Jackson and Wood [4].

The platform class hierarchy was refined to produce an alternate class hierarchy based on the army concept of battlefield operating systems. This illustrates the flexibility of the HLA simulation object model to provide more than one appropriate model of a simulation for military analysis and training. This flexibility in producing alternate class hierarchical structures can be used by the analyst to focus his data collection to that necessary to quantify his measures of effectiveness. The refined object class structure is shown inFigure 3.

The complete SOM may be found on the World Wide Web at the following URL: http://131.120.142.115/~buss/Larimer/JSOM5.8.omd

4.0 Comparison

We will now compare our object model development process with an 8-step sequence of activities proposed by Lutz [9]. Note that Lutz's methodology is proposed for SOM as well as FOM development, whereas ours is only applicable to the SOM development process. Table 2 summarizes Lutz's methodology.

Table 2: Summary of Lutz's Methodology

1. Determine Publishing Capabilities of Object/Interaction Classes.
2. Determine Subscription Requirements for Object/Interaction Classes.
3. Determine Publishing Capabilities for Attributes/Parameters
4. Determine Subscription Requirements for Attributes/Parameters.
5. Prepare Object Class Structure Table.
6. Prepare the Object Interaction Table.
7. Prepare Attribute/Parameter Table.
8. Prepare Object Model Template Extensions

Step 3 in Table 2 corresponds roughly with Step 2 of Table 1: Steps in the COM Process. The primary difference here is that it was first necessary to assign the available attributes to appropriate levels in the class hierarchy to form meaningful inheritance relationships among object classes. Additionally, identification of interaction parameters was deferred until the interaction classes were identified. For the reasons described previously, subscription requirements were not included in either the Janus COM or its SOM.

One can see that the two methodologies for producing an HLA object model are similar in many respects. The primary differences are in the development of the object class structure table and the sequence in which the attribute/parameter table is completed. The contrast between the two approaches is summarized in
Table 3. As Lutz points out, “It should be noted that this suggested sequence of development activities is not the only process that can lead to efficient and robust object model construction... many deviations from this process are possible which can lead to successful results” [9].

5.0 Role Of The Analyst

It is advantageous to include an analyst during SOM development, rather than only a programmer or implementer. As a minimum, the analyst should be an active member of any SOM development team. This is more important if the simulation model is to be used as an analysis tool.

The programmer or implementer can certainly produce an object model of a given model quickly and efficiently. However, it is the analyst who must use the model to quantify measures of effectiveness. Lutz points out that the model proponent has significant latitude in what is included in the SOM based the projected use of the model [9]. The analyst brings to the SOM development process an understanding of the kind of studies in which the model may be included, what measures of effectiveness the model may be expected to quantify, and therefore what is important to include in the SOM.

6.0 Summary

Implicit throughout this paper is an important point that is often lost in discussions of object-oriented design. Namely, the fact that a simulation’s representation via an Object Model is independent of the manner in which it is implemented. As we have demonstrated, it is possible to define an Object

Table 3: Comparison of Lutz and COM Methodologies

Step	Lutz SOM Development Process	COM Methodology
1	Determine Class Publishing Capabilities	Identify All Instantiable Objects
2	Determine Class Subscription Requirements	None
3	Determine Attribute/Parameter Publishing Capabilities	Identify All Attributes Available to Describe Objects
4	Determine Subscription Requirements for Attributes/Parameters	None
5	Prepare Object Class Structure Table	Build Class Hierarchy based on Common Attribute Object Groupings; Simultaneously Prepare Attribute portion of Attribute/Parameter Table
6	Prepare Object Interaction Table	Identify Interaction Parameters and Prepare Object Interaction Table; Simultaneously Prepare Parameter portion of Attribute/Parameter Table
7	Prepare Attribute/Parameter Table	Not Necessary
8	Prepare Object Model Template Extensions	Same
9	None	Reduce the Comprehensive Object Model to Produce a SOM Appropriate for Federation Needs

Model for Janus despite the fact that its implementation is completely procedural. A user of Janus through this Object Model interface would have no knowledge of, nor any need to know, Janus's implementation. Note that this is in fact a nice illustration of the object-oriented principle of encapsulation.

All legacy simulations should be considered as potential federation members. Developers and proponents of such simulations can proceed by developing Comprehensive Object Models, as we have done here for Janus. Since such a COM relies only on the simulation itself, it would not be necessary to obtain detailed information about a federation. Successful construction of a COM is primarily a function of the simulation's ability to be represented in abstract, object-oriented terms. In the case of Janus, the design of the database turned out to be a critical factor in its Object Model representation.

As mentioned previously, a SOM may easily be constructed from a COM by extracting those parts not necessary for interoperability. A COM being more general than a SOM, having a COM in hand makes construction of a SOM a fairly straightforward matter (although implementation for a particular SOM may not be entirely trivial). Models that come equipped with COMs could therefore be added to federations much more quickly than those without. A COM also gives flexibility for potential inclusions in multiple federations, since additional SOMs could be produced for different interoperability needs. Though potentially different, each valid SOM produced in this manner would necessarily be consistent with the underlying simulation, as well as any other Object Models so produced.

7.0 References

[1] *The Janus 3.X/VMS Model Software Programmer's Manual*, Contract No. DABT65-92-D-0002, November 1993.

[2] *HLA OMT Reference Version 1.0*, dated 15 Aug 96.

[3] *HLA OMT Extensions Reference Version 1.0*, dated 20 Aug 96.

[4] Jackson, Leroy A. and J. Ralph Wood, "Exploiting the High Level Architecture for Analysis in Advanced Distributed Simulation," 1997 Spring Simulation Interoperability Workshop, Orlando, FL, March 1997.

[5] *The Janus Version 6.3 Software User's Manual*, U.S. Army Simulation, Training, and Instrumentation Command, Orlando, FL, 1995.

[6] *The Janus Version 6.0 Database Manager's Manual*, U.S. Army Simulation, Training, and Instrumentation Command, Orlando, FL, 1995.

[7] Larimer, Larry, Arnold Buss, and Leroy Jackson, "Building a Simulation Object Model of a Legacy Simulation," 1997 Spring Simulation Interoperability Workshop, Orlando, FL, March 1997.

[8] Larimer, Larry R., Master of Science Thesis, Operations Research, *Building a Simulation Object Model of a Legacy Simulation (Draft)*, Naval Postgraduate School, June 1997.

[9] Lutz, Robert, "HLA Object Model Development: A Process View," Spring Simulation Interoperability Workshop, Orlando, FL, March 1997.

[10] *OMDT User's Guide, Alpha 1.0*, Developed by Aegis Research and Sponsored by The Defense Modeling and Simulation Office. 1996.

[11] Parish, Randall M. and Alvin D. Kellner, *Target Acquisition in Janus Army*, U.S. Army TRADOC Analysis Command, White Sands Missile Range, NM, Oct 92

[12] "JLINK - A Distributed Interactive Janus" by Major Maria C. Pate and Major Glen G. Roussos. http://131.120.57.3/jlink/JLINKphlanxArticle.html

[13] Roussos, Glen, "Attaining Interoperability between ModSAF and JANUS," 64th MORS Symposium, Fort Leavenworth, KS, June 1996.

8.0 Authors' Biographies

LEROY A. JACKSON, Major US Army, is an army artillery officer with twenty years of enlisted

and commissioned service. He graduated with a B.A. in Mathematics from Cameron University in 1990 and with an M.S. in Operations Research from the Naval Postgraduate School in 1995. He is currently assigned as an operations research analyst at the U.S. Army Training and Doctrine Command (TRADOC) Analysis Center (TRAC) Research Activities in Monterey, California and he continues his graduate studies at the Naval Postgraduate School.

ARNOLD H. BUSS is an Adjunct Professor of Operations Research at the Naval Postgraduate School. He received a BA in Psychology from Rutgers University, and MS in Systems and Industrial Engineering from the University of Arizona, and a Ph.D. in Operations Research from Cornell University. His research interests include simulation modeling and object-oriented software design. He is a member of INFORMS, MORS, and POMS.

LARRY R. LARIMER, Captain US Army, is an army infantry officer with fifteen years of enlisted and commissioned service. He graduated with a BS degree in Organizational Leadership from the United States Military Academy in 1986 and with a MS degree in Operations Research from the Naval Postgraduate School in 1997. He is currently an operations research analyst at the U.S. Army Training and Doctrine Command (TRADOC) Analysis Center, Fort Leavenworth. Captain Larimer's thesis research is the basis for this paper.

Annex A: A Portion of the Janus SOM Attribute/Parameter Table in OMDT Format

Object/Interaction	Attribute/Parameter	DataType	Cardinality	Units	Resolution	Accuracy	Accuracy Condition	Update Type	Update Condition	T/A	U/R
Base_Platform	Location	PlatformLocationType	1			perfect	always	Conditional	Timestep	N	UR
	Destination	PlatformLocationType	1			perfect	always	Conditional	User set	N	UR
	Speed	float	1	kph	0.1	perfect	always	Conditional	Terrain/Command Input	N	UR
	SystemNumber	short	1	N/A	Discrete	perfect	always	Static	None	N	UR
	SystemName	string	1	N/A	Discrete	perfect	always	Static	None	N	UR
	IDNumber	short	1	N/A	Integer	perfect	always	Conditional	None	N	R
	ElementSpace	short	1	Meters	Integer	perfect	always	Static	None	N	R
	CrewSize	short	1	Personnel	Integer	perfect	always	Conditional	Object destroyed	N	UR
	TrackWidth	float	1	Meters	0.01	perfect	always	Static	None	N	UR
	BellyWidth	float	1	Meters	0.01	perfect	always	Static	None	N	UR
	MagneticShadow	float	1	Meters	0.01	perfect	always	Static	None	N	UR
	ChemTransmitFactor	float	1		0.01	perfect	always	Static	None	N	R
	GraphicSymbolNumber	string	1	N/A	Integer	perfect	always	Static	None	N	R
	ClassSymbol	short	1	N/A	Integer	perfect	always	Static	None	N	R
	NormalInclusiveWeight	string	1	Pounds	Integer	perfect	always	Static	None	N	R
	NormalFullVolume	short	1	CuFt	Integer	perfect	always	Static	None	N	R
	Length	float	1	Meters	0.1	perfect	always	Static	None	N	R
	Width	any	1	Meters	0.1	perfect	always	Static	None	N	UR
	Height	float	1	Meters	0.1	perfect	always	Static	None	N	UR
	OpticalContrast	float	1	Unitless	0.01	perfect	always	Static	None	N	UR
	ThermalContrast_Exposed	any	1			perfect	always	Conditional		N	UR
	ThermalContrast_Defilade	any	1			perfect	always	Conditional		N	UR
	SensorType_Alternate	short	1	N/A	Integer	perfect	always	Static	None	N	R
	SensorType_Defilade	short	1	N/A	Integer	perfect	always	Static	None	N	R
	Status	boolean	1	N/A	Discrete	perfect	always	Conditional	Object destroyed	N	UR
	MOPP	boolean	1	N/A	Discrete	perfect	always	Conditional	User Input	N	UR
	SensorHeight	short	1	Meters	Integer	perfect	always	Static	None	N	R
	MaxVisibility	any	1			perfect	always	Conditional		N	UR
	Sensor_Primary	any	1			perfect	always	Conditional		N	UR
	Target_List	TargetListType	20	N/A	N/A	perfect	always	Conditional	Detection Occurs	N	UR
	ChemSelfRecDose	float	1	mg-min/cubic meter	0.1	perfect	always	Static	None	N	UR
	ChemCurrentDose	float	1	mg/cubic meters	0.1	perfect	always	Periodic	Time step-2 seconds	N	UR
	ChemIncapacitatSigma	float	1	mg/cubic meters	0.01	perfect	always	Static	None	N	UR
	ChemIncapacitMean	float	1	mg/cubic meters	0.001	perfect	always	Static	None	N	UR
	ChemDeathSigma	float	1	mg/cubic meters	0.01	perfect	always	Static	None	N	UR
	ChemDeathMean	float	1	mg/cubic meters	0.01	perfect	always	Static	None	N	UR
	ChemSelfRecogRespTime	float	1	Seconds	0.1	perfect	always	Static	None	N	UR
	ChemMaskTime	float	1	Seconds	0.1	perfect	always	Static	None	N	UR
	ChemCrewAlarmTime	float	1	Seconds	0.1	perfect	always	Static	None	N	UR
	ChemIncapacitRespTime	float	1	Seconds	0.1	perfect	always	Static	None	N	UR
	ChemExpirResponseTime	float	1	Seconds	0.1	perfect	always	Static	None	N	UR
	ChemDetectorWaitTime	float	1	Seconds	0.1	perfect	always	Static	None	N	UR
	ChemAlarmConcen	float	1	mg/cubic meters	0.01	perfect	always	Static	None	N	UR
	WorkRateStat	float	1	Watts	0.1	perfect	always	Static	None	N	UR
	WorkRateMove	float	1	Watts	0.1	perfect	always	Static	None	N	UR
	WorkRatePrep	float	1	Watts	0.1	perfect	always	Static	None	N	UR
	WorkRateFiring	float	1	Watts	0.1	perfect	always	Static	None	N	UR
	InitialCoreTemp	float	1	Degrees Fahrenheit	0.01	perfect	always	Static	None	N	UR
	MinCoreTemp	float	1	Degrees Fahrenheit	0.01	perfect	always	Static	None	N	UR
	MaxCoreTemp	float	1	Degrees Fahrenheit	0.01	perfect	always	Static	None	N	UR
	ClothInsulMOPP	float	1	N/A	0.01	perfect	always	Static	None	N	UR
	ClothInsulNoMOPP	float	1	N/A	0.01	perfect	always	Static	None	N	UR
	EvapImpedMOPP	float	1	N/A	0.01	perfect	always	Static	None	N	UR
	EvalImpedNoMOPP	float	1	N/A	0.01	perfect	always	Static	None	N	UR
	SpeedDegradMOPP	float	1	N/A	0.01	perfect	always	Static	None	N	UR
	FOVDegredMOPP	float	1	N/A	0.01	perfect	always	Static	None	N	UR
	HeatCurrentCoreTemp	float	1	Degrees Fahrenheit	0.01	perfect	always	Periodic	Time Step-1 minute	N	UR
	Side	any	1			perfect	always	Conditional		N	UR
	TaskForce	any	1			perfect	always	Conditional		N	UR
	Suppressed	boolean	1	N/A	N/A	perfect	always	Conditional	Engaged with direct fire	N	UR
Maneuver_Platform	WpnRelativeNum	short	1	N/A	Discrete	perfect	always	Conditional		N	UR
	WpnAbsoluteNum	any	1			perfect	always	Conditional		N	UR
	OrdnanceName	string	1	N/A	N/A	perfect	always	Conditional		N	UR
	BasicLoad	short	1	Each	Integer	perfect	always	Static		N	UR
	UploadTime	any	1			perfect	always	Conditional		N	UR
	WeaponToUse	any	1			perfect	always	Conditional		N	UR
	AmmoOnHand	short	1	Each	Integer	perfect	always	Conditional	Engagement Interaction	N	UR
	WeaponRange	float	1	Km	0.1	perfect	always	Static	None	N	R
	HoldFire	any	1			perfect	always	Conditional		N	UR
	FiringCategory	any	1			perfect	always	Conditional		N	UR
	TargetWpnMatch	WpnSelectionType	1+			perfect	always	Static	None	N	UR
	SmokeGrenBasicLoad	any	1			perfect	always	Conditional		N	UR